加微信：642945106 发送“ 赠送 ”领取赠送精品课程
发数字“ 2 ”获取众筹列表

04 | 案例：如何把流程化的测试脚本抽象为测试框架？

2020-02-07 陈磊

接口测试实战课 进入课程 >

讲述：陈磊
时长 14:18 大小 11.47M

你好，我是陈磊。

在上一节课中我和你强调了，用什么工具或代码解决测试问题并不重要，拥有接口测试思维才更重要。在今天的课程中，我会带你从零开始打造一个测试框架，建立你自己的技术体系。

在这里，我推荐你学习一门编程语言，以便你可以更加得心应手、个性化地改造你的测试框架或工具。在这节课中，我会以 Python 语言来展示我的代码示例，不过语言本身不是重点，你只需要了解这其中的逻辑与方法即可，同样的事情，你可以使用 Java、Go 等任何你喜欢的语言来完成。

当然，如果你想学习 Python 语言的话，我推荐你花一个周末的时间看看尹会生老师的“零基础学 Python”课程。

为什么要开发自己的测试框架？

之前，我们说到了用 Postman 来完成接口测试，但随着你的接口测试项目逐渐增加，你会发现越来越难以管理它的脚本，虽然测试工具导出的测试脚本也可以存放到代码仓库，但是，如果只是通过代码来查看是很难看懂的，你必须用原来的测试工具打开，才能更容易看懂原来的脚本做了什么样的操作。

同时，Postman 也有其自身的局限性，最重要的一点就是它支持的接口协议有限，如果你接到了一个它无法完成的接口类型的测试任务，就不得不再去寻找另一个工具。由于接口类型的多样和变化，你会有一堆工具需要维护，这无疑会提高你的学习成本和技术投入成本。

Postman 是如此，其他的工具也是如此，随着接口测试项目的增加，以及被测接口类型的增加，维护的难度会成指数级增长，所以，开发你自己的测试框架非常重要。

今天这节课，我就带你用 Python 3.7 来完成接口测试，并通过测试脚本的不断优化和封装，让你拥有一套完全适合你自己的接口测试框架。当然，我不会告诉你如何写出全部代码，我更想让你掌握的是，从不同的测试脚本抽象出一个测试框架的技巧和思路。

搭建测试框架，不要纠结于技术选型

在做接口测试脚本开发的技术选型上，我更建议你根据自己的技术实力和技术功底来选择，而不要以开发工程师的技术栈来选择。

这是因为，开发工程师和测试工程师关注的点，以及工作的交付目标是不同的。

> 对于任何一个开发工程师来说，他们主要的工作就是通过写代码实现产品需求或原型设计，他们会关心高并发、低消耗、分布式、多冗余，相对来说，也更加关注代码的性能和可靠性。
>
> 我们作为测试工程师，无论是使用自动化的接口测试，还是界面的手工测试，第一目标都是保障交付项目的质量，那些业务侧的表现，在大多数情况下不是我们关心的重点。

因此，开发工程师在开发技术栈上的使用频度、使用广度，都会远远高于我们，除非你本来就有对应的知识储备，否则不要强求炫技，为了提高工作效率，你只要使用自己熟悉的技术栈完成自动化接口测试就可以了。

这里我再强调一下，用什么技术栈来写代码，只是一种帮助你实现接口测试的手段，它不是你要交付的结果。所以你在搭建测试框架时，不要太纠结技术选型。

搭建前的准备工作

我相信现在你已经准备好，和我一起完成今天的内容了，但在开工之前，我要先把一些基础知识简单介绍给你。

我们今天会用到 Python 的一个第三方 HTTP 协议支持库 requests，它可以让我们在和 HTTP 协议打交道时更轻松；requests 项目的描述是“HTTP for Humans”，由此可见，这会是一个人见人爱的 HTTP 协议库。你可以通过下面这个命令，完成 requests 的安装：

复制代码

```
pip install requests
```

防止断更 请务必加
首发微信：1716143665

完成安装后，你就可以使用 requests 完成我用 Postman 完成的接口测试了。主要代码段我会在文章中给出，我会尽最大努力给你一个可以直接运行的代码段，这样，即使你看不懂也不用担心，你只要把这些代码复制到一个有 Python 运行环境的机器上，直接使用就可以了。

第一个接口的单接口测试脚本如下，我在代码中做了详细的注释，你既可以复制出去直接运行，也可以通过注释看懂代码的作用。这样，你就完成了一个无参数的、GET 访问的验证工作。

复制代码

```
# Python代码中引入requests库，引入后才可以在你的代码中使用对应的类以及成员函数
import requests
# 建立url_index的变量，存储战场的首页
url_index='http://127.0.0.1:12356/'
# 调用requests类的get方法，也就是HTTP的GET请求方式，访问了url_index存储的首页URL，返回结
response_index = requests.get(url_index)
# 存储返回的response_index对象的text属性存储了访问主页的response信息，通过下面打印出来
print('Response内容: '+response_index.text)
```

接下来，是第二个被测试的接口，它是登录接口，是以 POST 方式访问的，它需要通过 Body 传递 username 和 password 这两个参数，这两个参数都是字符串类型，字符长度不可以超过 10，并且不能为空。

你还记得在上节课中，我们一起用边界值法设计的测试用例吗？如果你忘记了，那么请你在本节课结束后，再回去看一下。这里你用下面的代码段，就可以完成第二个接口的单接口测试脚本了。

复制代码

```
# python代码中引入requests库，引入后才可以在你的代码中使用对应的类以及成员函数
import requests
# 建立url_login的变量，存储战场系统的登录URL
url_login = 'http://127.0.0.1:12356/login'
# username变量存储用户名参数
username = 'criss'
# password变量存储密码参数
password = 'criss'
# 拼凑body的参数
payload = 'username=' + username + '&password=' + password
# 调用requests类的post方法，也就是HTTP的POST请求方式，
# 访问了url_login，其中通过将payload赋值给data完成body传参
response_login = requests.post(url_login, data=payload)
# 存储返回的response_index对象的text属性存储了访问主页的response信息，通过下面打印出来
print('Response内容: ' + response_login.text)
```

无论你是不是看得懂上面的两段代码，你都能看出来，这其中有很多代码都是重叠在一起的，这两段代码的结构很相似，但又有明显的差异性。

开始打造一个测试框架

我想请你先思考这么一个问题，你在用 Postman 这类工具做接口测试时，除去你自己构建的访问路由和 Requsts 参数，其他的是不是就靠工具帮你处理完成了呢？

那么，我们接口测试的脚本，是不是也可以把一些公共的操作，抽象到一个文件中呢？这样你在写测试脚本时，通过拼凑路由、设计 Request 入参就可以完成接口测试了。在这样的思路之下，我们来一起改造一下刚刚的脚本。

第一步，你要建立一个叫做 common.py 的公共的方法类。下面我给出的这段注释详细的代码，就是类似我们使用 Postman 的公共方法的封装，它可以完成 HTTP 协议的 GET 请求或 POST 请求的验证，并且和你的业务无关。

复制代码

```
# 定义一个common的类，它的父类是object
class Common(object):
  # common的构造函数
  def __init__(self):
    # 被测系统的根路由
    self.url_root = 'http://127.0.0.1:12356'
  # 封装你自己的get请求，uri是访问路由，params是get请求的参数，如果没有默认为空
  def get(self, uri, params=''):
    # 拼凑访问地址
    url = self.url_root + uri + params
    # 通过get请求访问对应地址
    res = requests.get(url)
    # 返回request的Response结果，类型为requests的Response类型
    return res
  # 封装你自己的post方法，uri是访问路由，params是post请求需要传递的参数，如果没有参数这里
  def post(self, uri, params=''):
    # 拼凑访问地址
    url = self.url_root + uri
    if len(params) > 0:
      # 如果有参数，那么通过post方式访问对应的url，并将参数赋值给requests.post默认参数da
      # 返回request的Response结果，类型为requests的Response类型
      res = requests.post(url, data=params)
    else:
      # 如果无参数，访问方式如下
      # 返回request的Response结果，类型为requests的Response类型
      res = requests.post(url)
    return res
```

拼课微信：17161436665

接下来，用你自己的 Common 类，修改第一个接口的单接口测试脚本，就可以得到下面的代码了。

复制代码

```
# Python代码中引入requests库，引入后才可以在你的代码中使用对应的类以及成员函数
from common import Common
# 首页的路由
uri = '/'
# 实例化自己的Common
comm = Common()
#调用你自己在Common封装的get方法 ，返回结果存到了response_index中
response_index = comm.get(uri)
```

```
# 存储返回的response_index对象的text属性存储了访问主页的response信息，通过下面打印出来
print('Response内容: ' + response_index.text)
```

从这段代码中你可以看到，与前面对应的单接口测试脚本相比，代码的行数有明显的减少，这也能减少你很多的工作量，与此同时，如果你有任何关于 HTTP 协议的操作，都可以在 Common 类中进行修改和完善。

如果使用你自己刚刚建立的公共类（在我们内部有时候喜欢把它叫做轮子，这是源于一句俚语“不用重复造轮子”，因为 Common 类就是重复被各个检测代码使用的“轮子”）修改一下第二个接口的单接口测试脚本，代码就会变成下面这个样子：

复制代码

```
#登录页路由
uri = '/login'
# username变量存储用户名参数
username = 'criss'
# password变量存储密码参数
password = 'criss'
# 拼凑body的参数
payload = 'username=' + username + '&password=' + password
comm = Common()
response_login = comm.post(uri,params=payload)
print('Response内容: ' + response_login.text)
```

当你有一些更加复杂的脚本时，你会发现两次代码的变化会变得更明显，也更易读。

这就是那些曾经让你羡慕不已的框架诞生的过程，通过分析和观察你可以看到，原始的第一个接口的单接口测试脚本和第二个接口的单接口测试脚本，它们存在相同的部分，通过将这些相同的部分合并和抽象，就增加了代码的可读性和可维护性，也减少了脚本的开发量。通过这个方法，你就可以打造出一个属于自己的测试框架。

用你的框架完成多接口测试

上面我们仅仅进行了一小步的封装，就取得了很大的进步，在你写出越来越多的脚本后，你还会发现新的重叠部分，这时如果你能不断改进，最终就会得到完全适合你的测试框架，而且其中每一个类、每一个函数你都会非常熟悉，这样，碰到任何一个难解的问题时，你都有

能力通过修改你的框架来解决它，这样，这个框架实际上就变成了一个你在接口测试方面的工具箱了。

那么，怎么用我们刚刚一起搭建的测试框架，来完成多接口测试的业务逻辑测试呢？

不知道你是不是还记得在上节课中，我们讲到的 Battle 使用流程的测试用例，如果你没记起来，我先告诉你：“正确登录系统后，选择武器，与敌人决斗后杀死了敌人。”其他的，在本次课程结束后，你可以自己再去温习一下。

那么。使用我们一起封装的框架来完成上面的多接口测试后，就会得到下面的代码：

复制代码

```
# Python代码中引入requests库，引入后才可以在你的代码中使用对应的类以及成员函数
from common import Common
# 建立uri_index的变量，存储战场的首页路由
uri_index = '/'
# 实例化自己的Common
comm = Common()
#调用你自己在Common封装的get方法 ，返回结果存到了response_index中
response_index = comm.get(uri_index)
# 存储返回的response_index对象的text属性存储了访问主页的response信息，通过下面打印出来
print('Response内容: ' + response_index.text)
# uri_login存储战场的登录
uri_login = '/login'
# username变量存储用户名参数
username = 'criss'
# password变量存储密码参数
password = 'criss'
# 拼凑body的参数
payload = 'username=' + username + '&password=' + password
comm = Common()
response_login = comm.post(uri_login,params=payload)
print('Response内容: ' + response_login.text)
# uri_selectEq存储战场的选择武器
uri_selectEq = '/selectEq'
# 武器编号变量存储用户名参数
equipmentid = '10003'
# 拼凑body的参数
payload = 'equipmentid=' + equipmentid
comm = Common()
response_selectEq = comm.post(uri_selectEq,params=payload)
print('Response内容: ' + response_selectEq.text)
# uri_kill存储战场的选择武器
uri_kill = '/kill'
# 武器编号变量存储用户名参数
enemyid = '20001'
```

```
# 拼凑body的参数
payload = 'enemyid=' + enemyid+"&equipmentid="+equipmentid
comm = Common()
response_kill = comm.post(uri_kill,params=payload)
print('Response内容: ' + response_kill.text)
```

上面的代码有点长，但你先不要有抵触的心理，每一个代码行的注释我都写得很清楚。然而我并不是想让你知道，上面那么多类似蝌蚪文的代码都是干什么的，我是想让你看看上面的代码中，是否有可以用前面“抽象和封装重复代码的方法”进行优化的地方。

你可以看到，上面的代码大量重复了你自己写的通用类的调用，这个其实是可以合成一个的；同时，你再观察一下我们一起写的 Common 类，你会发现有一个 self.url_root = ‘http://127.0.0.1:12356’ ，如果这里这样写，你的 Common 就只能用来测试我们这个小系统了，除非你每次都去修改框架。

但是，任何一个框架的维护者，都不希望框架和具体逻辑强相关，因此这也是一个优化点，那么将上面的内容都修改后，代码就会变成下面这个样子：

复制代码

```
# Python代码中引入requests库，引入后才可以在你的代码中使用对应的类以及成员函数
from common import Common
# 建立uri_index的变量，存储战场的首页路由
uri_index = '/'
# 实例化自己的Common
comm = Common('http://127.0.0.1:12356')
#调用你自己在Common封装的get方法 ，返回结果存到了response_index中
response_index = comm.get(uri_index)
# 存储返回的response_index对象的text属性存储了访问主页的response信息，通过下面打印出来
print('Response内容: ' + response_index.text)
# uri_login存储战场的登录
uri_login = '/login'
# username变量存储用户名参数
username = 'criss'
# password变量存储密码参数
password = 'criss'
# 拼凑body的参数
payload = 'username=' + username + '&password=' + password
response_login = comm.post(uri_login,params=payload)
print('Response内容: ' + response_login.text)
# uri_selectEq存储战场的选择武器
uri_selectEq = '/selectEq'
# 武器编号变量存储用户名参数
equipmentid = '10003'
```

```
# 拼凑body的参数
payload = 'equipmentid=' + equipmentid
response_selectEq = comm.post(uri_selectEq,params=payload)
print('Response内容: ' + response_selectEq.text)
# uri_kill存储战场的选择武器
uri_kill = '/kill'
# 武器编号变量存储用户名参数
enemyid = '20001'
# 拼凑body的参数
payload = 'enemyid=' + enemyid+"&equipmentid="+equipmentid
response_kill = comm.post(uri_kill,params=payload)
print('Response内容: ' + response_kill.text)
是不是比上一个节省了很多代码，同时也看的更加的易读了，那么我们封住好的Common就变成了如下的样子

# 定义一个common的类，它的父类是object

class Common(object):

# common的构造函数

def __init__(self,url_root):

# 被测系统的跟路由

self.url_root = url_root



# 封装你自己的get请求，uri是访问路由，params是get请求的参数，如果没有默认为空

def get(self, uri, params=''):

# 拼凑访问地址

url = self.url_root + uri + params

# 通过get请求访问对应地址

res = requests.get(url)
```

```
77
78  # 返回request的Response结果，类型为requests的Response类型
79
80
81
82
83
84  return res
85
86
87  # 封装你自己的post方法，uri是访问路由，params是post请求需要传递的参数，如果没有参数这里为空
88
89
90
91  def post(self, uri, params=''):
92
93  # 拼凑访问地址
94
95
96
97  url = self.url_root + uri
98
99
100 if len(params) > 0:
101
102 # 如果有参数，那么通过post方式访问对应的url，并将参数赋值给requests.post默认参数data
103
104
105 # 返回request的Response结果，类型为requests的Response类型
106
107
108 res = requests.post(url, data=params)
109
110
111 else:
112
113
114 # 如果无参数，访问方式如下
115
116
117 # 返回request的Response结果，类型为requests的Response类型
118
119
120 res = req
121
```

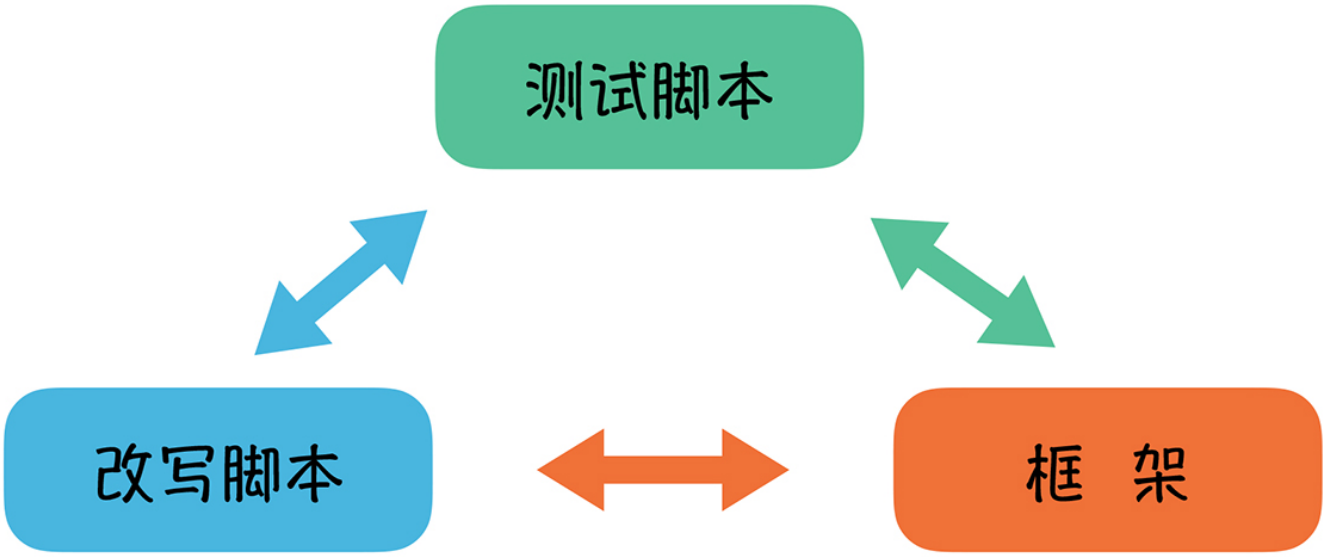
你可以看到，在上面这段代码中，我主要是让我们 Common 类的构造函数接受了一个变量，这个变量就是被测系统的根路由。这样是不是就比上一个代码段节省了很多代码，同时也更加易读了？那么我们封装好的 Common 就变成了下面这个样子：

复制代码

```
# 定义一个common的类，它的父类是object
class Common(object):
  # common的构造函数
  def __init__(self,url_root):
    # 被测系统的跟路由
    self.url_root = url_root
  # 封装你自己的get请求，uri是访问路由，params是get请求的参数，如果没有默认为空
  def get(self, uri, params=''):
    # 拼凑访问地址
    url = self.url_root + uri + params
    # 通过get请求访问对应地址
    res = requests.get(url)
    # 返回request的Response结果，类型为requests的Response类型
    return res
  # 封装你自己的post方法，uri是访问路由，params是post请求需要传递的参数，如果没有参数这里
  def post(self, uri, params=''):
    # 拼凑访问地址
    url = self.url_root + uri
    if len(params) > 0:
      # 如果有参数，那么通过post方式访问对应的url，并将参数赋值给requests.post默认参数da
      # 返回request的Response结果，类型为requests的Response类型
      res = requests.post(url, data=params)
    else:
      # 如果无参数，访问方式如下
      # 返回request的Response结果，类型为requests的Response类型
      res = requests.post(url)
      return res
```

通过改造 Common 类的构造函数，这个类已经变成一个通用类了，无论是哪一个项目的接口测试，都可以使用它来完成 HTTP 协议的接口验证了。

我相信现在你已经掌握了测试框架的形成过程，就如下图所示，测试框架的形成是在撰写大量测试脚本的过程中不断抽象封装出来的，然后，再用这个不断完善的框架，改写原有的测试脚本。循环往复这个过程，你就会慢慢获得一个独一无二的、又完全适合你工作的接口测试框架。



其实到这里，我们上面说的只能算是一个调试代码，还不能算是一个测试框架。上面这些代码所有的返回值都打印到控制台后，为了完成接口测试，你需要时时刻刻看着控制台，这还不能算是自动化，只能说是一个辅助小工具。

在这里，你应该让全部测试结果都存储到测试报告里面，同时通过一个测试驱动框架来完成各个模块的驱动，这也是为什么你在学习任何一种框架的时候，总会遇见类似 Java 的 JUnit、Python 的 Unittest 的原因，因此，上面的 Common 类还需要和 Python 的 unittest 一起使用，才算是一个完美的测试框架。

至于你自己的 Common 类怎么和测试驱动框架相结合，这部分内容就留给你在未来的接口测试工作中，自己去学习并完成了。

总结

今天，我们一起学习了一个测试框架的诞生过程。测试框架就是在你测试脚本中不断抽象和封装得来的。今天我们课程的内容充斥着各种代码，如果你的代码基础稍微比较薄弱，并没有完全记住上面的内容，那么我希望你记住从测试脚本到测试框架的转化过程：

1. 不断撰写测试脚本，所有的抽象和封装都是站在已有的测试脚本基础之上的；
2. 多观察已经写好的测试脚本，找出其中的重叠部分，最后完成封装；

3. 以上两步是一个不断循环又循序渐进的过程，你要在你的工作中始终保持思考和警惕，发现重复马上进行框架封装。

最后我想和你强调的是，测试框架的封装和抽象过程并不是一蹴而就的，它是靠一点一点的积累得来的，因此，你要通过自己的实践，慢慢积累和完善你的测试框架，而不要妄想一次就能有一个完善的测试框架。我相信，当你通过写脚本完成整个项目的接口测试后，你一定会得到一个完美的测试框架。

思考题

在我讲的最后一个多接口测试脚本，其实也并不是最完美的修改，你能提出更好的修改意见吗？如果它可以抽取到你的框架中，那么是完成一个什么样任务的类或者函数呢？

我是陈磊，欢迎你在留言区留言分享你的观点，如果这篇文章让你有新的启发，也欢迎你把文章分享给你的朋友，我们一起探讨和学习。

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇　03 | 思维方式：用一个案例彻底理解接口测试的关键逻辑

下一篇　05 | 案例：测试框架如何才能支持RESTful风格的接口？

精选留言 (10)

写留言

happychap
2020-02-08

有个观点与老师不同，就是在编程语言的选择上，针对新手而言，选择与研发团队相同的编程语言（java,python,js,go）个人觉得会更好一些，一方面有研发团队作为依靠，另一方面在接入ci时，可以复用研发团队的ci接入技术栈。再有就是能复用面向测试的底层开源库就先复用它们，把更多的精力放在测试领域功能封装上，如：入参的处理，出参的传递，接口请求结果的抽取等方面。...

展开

作者回复: 您好，看了您的留言，你的观点也有道理，我不推荐按照rd的技术栈选择是因为rd的技术栈入门门槛高，ci，cd流水线也不依赖rd技术栈，因此我建议在基础薄弱的小伙伴选择上以简单的开始。

1 3

杨亚玲
2020-02-08

有下面这样的错误：
print('Response内容：' + response_login.text)
AttributeError: 'NoneType' object has no attribute 'text'

作者回复: 您好，我在我的电脑上没有发现有类似保存，您方便把全部代码贴出来看一下吗。

1 1

Kino_testdevelopment
2020-02-10

老师有相关的测试框架可以给我使用学习的么？

展开

作者回复: 目前没推荐现有框架，更提倡自己积累自己的框架

安安
2020-02-10

老师，您好，想请教一下：一个项目的接口测试建议是直接用代码完成还是用工具，类似postman呢？测试同学有代码基础。

展开

作者回复: 谢谢，这个选择依照你团队的技术栈选择，如果代码技术好点，优先选择代码会好一点

1

苦行僧
2020-02-09

我司就是直接购买一个商业api测试产品

展开

作者回复: 谢谢，我还没有见过商业化的产品，方便留下产品名字吗

7F
2020-02-09

有一个Python业界很获得接口测试框架，HTTPrunner，老师你怎么看这个框架。

作者回复: 每个框架都有它的优势和劣势，HTTPRunner我还没听说，有时间我去学习一下。

陈优雅
2020-02-09

单一接口测试代码可以用postman直接生成

展开

作者回复: 谢谢，后续课程也就有相关讲解

roychris
2020-02-08

是不是可以按照业务流程封装一个大类，从头到尾去执行不同的分支?

作者回复: 谢谢您，您可以按照您的想法去实践，我很提倡大家去试错，实践出真知，我也特别期望您的想法落地后的效果和感悟😬

Geek_a57228
2020-02-07

老师，看你的脚步，写公共方法的那一段是否需要加上import requests呢？不然后面调用requests包时，会有问题吧，刚学Python代码，所以问题比较初级

展开

作者回复: 嗯，是的谢谢指出。

高源

2020-02-07

上面的 Common 类还需要和 Python 的 unittest 一起使用，才算是一个完美的测试框架。这块老师简单补充下我想知道具体怎么结合的谢谢

展开

作者回复: 您好，我没有写是因为网络上unittest教程特别多，就不重复在这里讲解了。你可以自行查找一下

1